

AFRL-IF-RS-TR-2002-61

Final Technical Report

April 2002



COMPOSABILITY, PROVABILITY, REUSABILITY (CPR) FOR SURVIVABILITY

Kestrel Institute

**Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. E017**

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

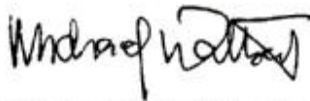
AFRL-IF-RS-TR-2002-61 has been reviewed and is approved for publication.

APPROVED:



NANCY A. ROBERTS
Project Engineer

FOR THE DIRECTOR:



MICHAEL TALBERT, Maj., USAF, Technical Advisor
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 074-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE APRIL 2002		3. REPORT TYPE AND DATES COVERED Final Oct 96 – Dec 99
4. TITLE AND SUBTITLE COMPOSABILITY, PROVABILITY, REUSABILITY (CPR) FOR SURVIVABILITY			5. FUNDING NUMBERS C - F30602-96-C-0363 PE - 62301E PR - E017 TA - 04 WU - 02	
6. AUTHOR(S) Allen Goldberg				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Kestrel Institute 3260 Hillview Avenue Palo Alto California 94304			8. PERFORMING ORGANIZATION REPORT NUMBER N/A	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency AFRL/IFTD 3701 North Fairfax Drive 525 Brooks Road Arlington Virginia 22203-1714 Rome New York 13441-4505			10. SPONSORING / MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2002-61	
11. SUPPLEMENTARY NOTES AFRL Project Engineer: Nancy A. Roberts/IFTD/(315) 330-3566				
12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.				12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 Words) The goal of this effort Composability, Provability, Reusability (CPR) for Survivability is to address the problem of composition of survivable systems. The particular objective of this project is to construct a formal specification of the Java Virtual Machine (JVM) bytecode loader and verifier, and from that specification formally derive a provably-correct implementation. The specification and program development is being carried out using Kestral's Specware System. The security of Java applications depend on type safety and related properties enforced by bytecode verification. Serious Java security flaws have been traced to errors in Sun's Java bytecode verifier and loader. A formal specification will serve as a reference document for the construction of new JVM implementations for just-in-time compilers, web browsers, smart cards, etc. The desired safety and security properties of the verifier will be proved as putative properties of the formal specification. The formally-derived implementation can be used as a test oracle to test implementations, or may be incorporated directly into a JVM implementation.				
14. SUBJECT TERMS Java Virtual Machine, Formal Methods, Java Security, Java Formal Specification			15. NUMBER OF PAGES 9	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Table of Contents

1. Introduction	1
1.1. Overview	1
1.2. Use of Specware	1
1.3. Accomplishments	1
1.4. Results	2
Appendix A	4

List of Figures

Figure 1 - A.java	2
Figure 2 - B.java (WRONG)	3
Figure 3 - B.java (CORRECTED)	3

1. Introduction

1.1. Overview

Java™ is perhaps the first programming system that is both widely used, and conceptually clean enough to undergo a formal analysis. This represents a unique opportunity for formal methods to make an impact on computing practice. Java is finding application in security critical applications especially Internet programming and mobile code applications.

The work on this contract focused on low-level Java Security. The Java compiler translates Java class definitions into platform-independent target language known as the Java Virtual Machine. The *Java Virtual Machine (JVM)* is a type-safe, stack-oriented abstract machine. *JVM* code (also called bytecode), not Java source is transmitted when an “applet” is sent over the Internet and remotely executed. Because the transmitted code cannot be trusted to be the unmodified output of a correct Java (or other language) compiler, the code must be checked for consistency either prior to execution or by run-time checking. Thus security issues in Java are focused on the JVM. This consistency, which involves type safety and other issues, is the subject of our investigation.

A secure system must have a secure foundation. This foundation is referred to as low-level security. In the JVM, language mechanisms are used to insure low-level security. These mechanisms insure that mechanisms for attacking a system such as buffer overflow attacks cannot be launched against the JVM. With that foundation, high-level security can be provided by a security manager that regulates access to system resources and provides a level of separation between independently executing threads. Kestrel’s research program will continue by formalizing the JVM security manager.

Kestrel has formalized in mathematical notation or Specware specifications the bytecode verifier and the class loader, two key components of the JVM runtime that have both subtle semantics and play an essential role in the security of the JVM. Indeed this investigation has exposed bugs in the Sun implementation of the JVM that leads to a failure of type safety. This bug exploits the interplay of the bytecode verifier, the class loader, and class resolution.

1.2. Use of Specware

Specware is a prototype system developed at Kestrel Institute for the formal development of executable code from specifications. Kestrel used the Specware system to generate code for the bytecode verifier from a formal specification. It was one of the largest applications of Specware to date, and help to shape development of the system.

1.3. Accomplishments

Note that funding from the NSA also contributed to these results.

- Kestrel has produced the first and to this date only full formalization of the bytecode verifier that includes all of its functionality.
- This formalization has lead to design improvements that we are proposing to Sun. These improvements will allow faster and more flexible loading of classes. In

particular the bytecode verifier need not load certain specifications to insure type safety.

- This formalization is written in such a way as to allow extensions of the verifier to be specified in a modular way. Kestrel anticipates that such extension will be developed to add new security-related functionality to the JVM.
- Kestrel has written Specware specifications for a subset of these formalizations and has refined these specifications to executable code.
- Kestrel has formalized the essential nature of the JVM class loader and is able to prove a type safety result for the combination of the bytecode verifier and class loader.

1.4. Results

This effort is detailed in a series of technical papers listed in appendix A. Most of the papers describe the specification of the JVM in Specware. The [4] paper lists some of the problems in the JDK 1.2.2 found using the derived bytecode verifier. A simple example showing the benefits of using a formally derived bytecode verifier developed using Specware as opposed to a handwritten bytecode verifier is illustrated below.

In this example, there are two classes: A and B. Figure 1 shows that Class A has two simple functions(change and m) defined for integers. Class B, shown in Figure 2, just simply extends Class A to include versions of those functions for float.

```
Class A
{
  int i;
  A(){ }

  void change(int j){
    i = j;
  }

  int m(int j){
    return i + j; }
}
```

Figure 1 - A.java

```

Class B extends A
{
    float f ;
    B() { }

    void change(float f1){
        f = f1; }

    float m(float f1){
        return f + f1; }
}

```

Figure 2 - B.java (WRONG)

```

Class B extends A
{
    float f = 0;
    B() { }

    void change(float f1){
        f = f1; }

    float m(float f1){
        return f + f1; }
}

```

Figure 3 - B.java (CORRECTED)

These two files both compile ok in JDK 1.2.2. However if one was to disassemble B.class, the statement *return f + f1* would be using an integer add instead of a floating point add and thus could result in an obscure error. The Specware bytecode verifier would compile B.java and give an error. Figure 3 illustrates a change that would make both the JDK1.2.2 and Specware bytecode verifier versions so that the B.java file compiles correctly.

Appendix A

1. [A Specification of Java Loading and Bytecode Verification](#). Allen Goldberg. *Proceedings, 5th ACM Conference on Computer and Communications Security*, San Francisco, October 1998. *Kestrel Institute Technical Report KES.U.97.1*, December 1997.
2. [A Formal Specification of Java Class Loading](#). Z. Qian, A. Goldberg, and A. Coglio. *Proc. 15th ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'00)*. ACM SIGPLAN Notices, v. 35(10), October 2000, pp. 325-336. Also *Kestrel Institute Technical Report KES.U.99.6*, October 1999.
3. [Type Safety in the JVM: Some Problems in JDK 1.2.2 and Proposed Solutions](#). Alessandro Coglio and Allen Goldberg. *Proc. 2nd ECOOP Workshop on Formal Techniques for Java Programs*. June 2000. *Kestrel Institute Technical Report KES.U.00.3*, June 2000.
4. [Towards a Provably-Correct Implementation of the JVM Bytecode Verifier](#). Alessandro Coglio, Allen Goldberg, and Zhenyu Qian. *Proceedings of the OOPSLA '98 Workshop on the Formal Underpinnings of Java*, Vancouver, B.C., October 1998. *Kestrel Institute Technical Report KES.U.98.5*, August 1998.
5. [A Formal Specification of a Large Subset of Java\(tm\) Virtual Machine Instructions for Objects, Methods and Subroutines](#). Zhenyu Qian. *Formal Syntax and Semantics of Java(TM)*. Alves-Foss, J. (Ed.), Springer Verlag LNCS, 1998. *Kestrel Institute Technical Report KES.U.98.4*, August 1998.
6. [Standard Fixpoint Iteration for Java Bytecode Verification](#). Zhenyu Qian. *ACM Transactions on Programming Languages and Systems*, Vol. 22(4), July 2000, pp. 638-672. *Kestrel Institute Technical Report KES.U.00.6*, July 2000.